On the Simplification of Neural Network Architectures for Predictive Process Monitoring

Amaan Ansari[1,2], Lukas Kirchdorfer[1,2], and Raheleh Hadian[1]

[1] SAP Signavio, Walldorf, Germany
[2] University of Mannheim, Data and Web Science Group, Mannheim, Germany

Abstract. Predictive Process Monitoring (PPM) aims to forecast the future behavior of ongoing process instances using historical event data, enabling proactive decision-making. While recent advances rely heavily on deep learning models such as LSTMs and Transformers, their high computational cost hinders practical adoption. Prior work has explored data reduction techniques and alternative feature encodings, but the effect of simplifying model architectures themselves remains underexplored. In this paper, we analyze how reducing model complexity—both in terms of parameter count and architectural depth—impacts predictive performance, using two established PPM approaches. Across five diverse event logs, we show that shrinking the Transformer model by 85% results in only a 2–3% drop in performance across various PPM tasks, while the LSTM proves slightly more sensitive, particularly for waiting time prediction. Overall, our findings suggest that substantial model simplification can preserve predictive accuracy, paving the way for more efficient and scalable PPM solutions.

Key words: predictive process monitoring, deep learning, process mining

1 Introduction

Predictive Process Monitoring (PPM) has become a central topic in Business Process Management (BPM), shifting the focus from retrospective analysis to forecasting future aspects of ongoing process instances. By anticipating properties such as the next activity, the next resource, upcoming timestamps, remaining time, or the final outcome, PPM enables organizations to make proactive decisions, optimize resource allocation, and mitigate potential risks [1, 2, 3].

Recent advances in PPM have been largely driven by deep learning, with Recurrent Neural Networks (RNNs) and in particular Long Short-Term Memory (LSTM) networks proving effective in modeling the sequential nature of event logs [4, 5]. More recently, Transformer-based models, originally developed for Natural Language Processing (NLP), have been successfully adapted for PPM. Models such as ProcessTransformer [6] and MTLFormer [7] leverage self-attention mechanisms to capture long-range dependencies in event sequences—an area where traditional RNNs often struggle.

Despite their predictive power, these deep learning models are computationally intensive, with large numbers of parameters and high training costs. This complexity presents a barrier to practical deployment, particularly in industrial settings where computational resources are limited. For instance, process mining vendors like SAP Signavio, Celonis, or Apromore serve a large number of diverse clients across many domains, each with unique processes. Delivering PPM capabilities at scale would require training and maintaining separate models for each process and customer, leading to high costs and limited scalability.

In this paper, we investigate whether pruning of state-of-the-art deep learning architectures can deliver competitive performance while significantly reducing computational overhead. We focus on streamlined variants of the LSTM-based model proposed by Camargo et al. [5] and the Transformer-based MTLFormer introduced by Wang et al. [7], evaluating their potential as lightweight alternatives for PPM in resource-constrained environments. Through our experimental investigation, we present the following key findings:

- We found that our simplified Transformer models achieve prediction performance similar to the full MTLFormer architecture, being on par or only slightly worse on average across several PPM tasks, but with a significant reduction in model parameters of around 85%. This suggests that substantial parameter reduction in Transformer models is possible without a proportional loss in performance for PPM tasks.
- In contrast, simplifying the LSTM architecture to a similar degree in terms of model parameters often leads to a more pronounced performance degradation, showing a performance drop of 3-13% on average across the core PPM tasks.
- Besides these, our experiments revealed a few more interesting findings, such as the respective strengths and weaknesses of LSTM and Transformer models for the mentioned PPM tasks.

The remainder of this paper is structured as follows: Section 2 provides the necessary background on PPM and discusses related work. Then, Section 3 introduces the required concepts and presents the employed architectures and the training procedure, followed by our evaluation in Section 4. Finally, we conclude the work in Section 5.

2 Background and Related Work

In this section, we provide the necessary background on PPM and survey existing efforts to improve the computational efficiency of these predictive models.

2.1 Predictive Process Monitoring

PPM has emerged as a promising area within process mining, offering a wide range of business applications. Early research in this domain focused primarily on predicting process outcomes, particularly in terms of duration and successful

completion. To this end, a variety of techniques have been proposed, e.g., based on Hidden Markov Models [8], finite state machines [9], or stochastic Petri nets [10]. In addition, classical machine learning approaches—such as random forests and support vector machines—have been adapted to predict process outcomes using handcrafted features derived from event histories [11].

With the advent of deep learning, and in particular LSTM networks, the focus shifted toward automated feature learning from large-scale, sequential event logs. Early work by Evermann et al. [12] demonstrated the effectiveness of shallow LSTM models combined with embedding techniques for categorical variables in predicting the next event. Building on this, Tax et al. [4] used a similar LSTM architecture with one-hot encodings to predict the next activity, its corresponding timestamp, the remaining time, and the full process suffix. Camargo et al. [5] extended this approach by combining multiple LSTM components to support both categorical and numerical features.

Beyond LSTM-based models, Transformer architectures have gained traction in PPM. Bukhsh et al. [6] introduced the ProcessTransformer to jointly predict the next activity, the next timestamp, and the remaining time. Wang et al. [7] further advanced this line of work with the MTLFormer, a multi-task learning model that addresses all three prediction tasks within a unified framework. Graph Transformer architectures have also been explored, particularly for the task of remaining time prediction [13].

More recent research has leveraged PPM models also in the context of process simulation, either as a component of the simulation model [5] or for the evaluation of the simulation model's quality [14].

2.2 Towards More Efficient PPM Models

The deep learning models discussed above, while effective, often demand substantial computational resources and long training times, posing significant challenges for large-scale deployment in resource-constrained environments. Several studies in the PPM community have approached this issue from different perspectives. One line of work focuses on reducing the size of the training dataset. For example, Sani et al. [15] propose a sampling strategy to select an informative subset of the data for model training. Pauwels et al. [16] address the problem by introducing incremental learning techniques that update neural networks with new data, thereby avoiding full retraining. Another strategy involves exploring alternative model architectures to improve efficiency. Weytjens and De Weerdt [17] demonstrate that Convolutional Neural Networks (CNNs) can offer both faster training and improved accuracy compared to LSTMs. Additionally, modifications to sequence encoding techniques have been proposed to further enhance training efficiency. Specifically, Roider et al. [18] propose trace encoding as an alternative to prefix sequence encoding.

While these approaches have shown promise in reducing training time and resource usage, none have explicitly investigated whether simplifying the architecture itself, by reducing the number of parameters and overall model size,

can yield efficient PPM solutions without compromising predictive performance. This gap motivates our investigation into lightweight model variants for PPM.

3 Methodology

This section outlines the methodology of our study. We start by introducing the key concepts and defining the various PPM tasks. We then describe the MTLFormer [7], the LSTM model [5], and our simplified variants of these architectures. Finally, we detail the training procedure.

3.1 Preliminaries and Problem Definition

We begin by defining core concepts such as event log, trace, and prefix before formally defining the PPM tasks.

Event log, traces, prefixes. We assume that process execution data is stored in an event log $\mathcal{E}$, defined as a finite set of traces. Each trace $\sigma = \langle e_1, e_2, ..., e_n \rangle$ represents the ordered sequence of events for a single process instance (case). We define an event $e = (c, a, r, t_s, t_e)$ as a tuple consisting of a case ID c, an activity a, a role r, a start timestamp t_s, and an end timestamp t_e. For each element $l \in \{c, a, r, t_s, t_e\}$ of an event e, we define a projection function π_L such that $\pi_L(e)$ returns that element (e.g. $\pi_A(e) = a$). PPM is typically formulated as a classification or regression problem and begins with a feature extraction step. This involves generating prefixes of varying lengths from each completed trace to represent different execution stages. Formally, for a given trace σ, we define its prefix of length $k \in [1, n-1]$ as $p^k(\sigma) = \langle e_1, ..., e_k \rangle$, representing a partial execution up to the k-th event of a trace with in total n events. Each prefix is then encoded into a feature vector $\boldsymbol{x} \in \mathcal{X}$ and associated with one or more target values $y_t \in \mathcal{Y}$, each corresponding to a specific PPM task.

Definition of PPM tasks. We consider five common PPM tasks, which we formally introduce in the following. Each task takes as input an event prefix $p^k(\sigma)$ of a trace σ.

1. **Next activity prediction**: Defined as $\Theta_a(p^k(\sigma)) = \pi_A(e_{k+1})$, predicting the activity of the next event.
2. **Next role prediction**: Defined as $\Theta_r(p^k(\sigma)) = \pi_R(e_{k+1})$, predicting the role responsible for the next event.
3. **Next event duration prediction**: Defined as $\Theta_d(p^k(\sigma)) = \pi_{T_e}(e_{k+1}) - \pi_{T_s}(e_{k+1})$, predicting the difference between the next event's end and start timestamps.
4. **Next waiting time prediction**: Defined as $\Theta_{wt}(p^k(\sigma)) = \pi_{T_s}(e_{k+1}) - \pi_{T_e}(e_k)$, predicting the difference between the next event's start timestamp and the current event's end timestamp.
5. **Remaining time prediction**: Defined as $\Theta_{rt}(p^k(\sigma)) = \pi_{T_e}(e_n) - \pi_{T_e}(e_k)$, predicting the difference between the timestamp of the case's last event and the current event's end timestamp.

3.2 Architectures

We evaluate five architectures spanning both Transformer- and LSTM-based designs: the original MTLFormer [7], our light variant of the MTLFormer, our model with a single transformer encoder ($\text{Transformer}_{\text{simple}}$), instead of multiple, the original LSTM model proposed by Camargo et al. [5], and our light variant of this.

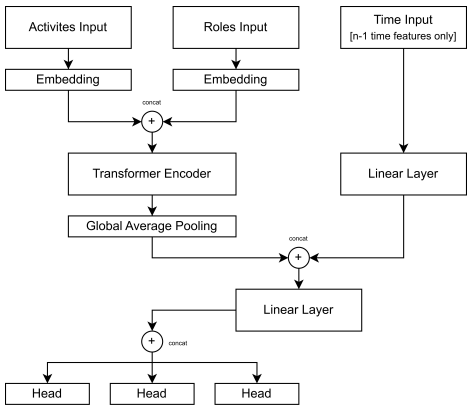


Fig. 1: Our $\text{Transformer}_{\text{simple}}$ architecture

Transformer Models. We begin by describing the three Transformer models:

- *MTLFormer* uses five parallel Transformer streams, two of which are dedicated to activity labels, two to role labels, and one to temporal context. We denote these streams as *activity stream 1*, *activity stream 2*, *role stream 1*, *role stream 2*, and the *temporal stream*. Each of these streams ingests its corresponding feature set (activity, role, or temporal), passes it through a Transformer encoder, and then applies average pooling to yield a fixed-length embedding. A Transformer encoder stacks multiple layers, each performing multi-head self-attention (MHA). The embeddings from activity stream 1, role stream 1, and the temporal stream are concatenated and projected. This projection is then concatenated with the embeddings from activity stream 2 and role stream 2. The concatenated representation is then fed into three deep multi-layer perceptrons (MLPs) acting as prediction heads, which forecast the next activity, the next role, and time-related values. Originally, the MTLFormer [7] predicts solely the next activity, the next timestamp, and the remaining time. However, we have extended both the inputs and the output heads to incorporate role prediction and waiting-time estimation, without altering the core architecture.
- $\textit{MTLFormer}_{\textit{light}}$ consists of a backbone and three prediction heads. It preserves the original five-stream Transformer backbone architecture of the MTLFormer while simplifying each prediction head to a single linear layer,

instead of using MLPs. In addition, the number of parameters is reduced by shrinking the hyperparameters of the backbone. These include, but not limited to, the number of heads, the input embedding size and the feed-forward dimensions of the MHA. This modification is designed to reduce both parameter count and inference time without altering the backbone architecture.
- *Transformer*$_{simple}$ (see Figure 1) simplifies MTLFormer$_{\text{light}}$ even further by removing its five parallel encoder streams and processing activity and role tokens within a single Transformer encoder. This yields a single Transformer encoder as the backbone, with three prediction heads, each of which consists of a single linear layer. The tokens are embedded, concatenated channel-wise, and passed through the Transformer encoder. The resulting feature maps are average-pooled and combined with a linear projection of the temporal features from the final input position. A shallow linear layer with ReLU activation function then feeds three single-layer heads for next activity, next role, and time predictions. This single-encoder design offers a substantially leaner architecture, enabling a direct comparison with MTLFormer$_{\text{light}}$.

LSTM Models. Next, we describe the two LSTM models:
- *LSTM*—with which we refer to the model proposed by Camargo et al. [5]—embeds activity, role, and temporal features, concatenates them, and feeds the sequence into a shared LSTM layer with batch-normalisation and dropout. This shared LSTM layer acts as the backbone. Afterwards, the last hidden state of the shared LSTM layer is routed into three parallel, task-specific heads, one each for activity, role, and time. Each head consists of a single LSTM layer and a small MLP.
- *LSTM*$_{light}$ retains the shared LSTM layer as backbone, but changes the prediction heads. Instead of using a single LSTM layer and a small MLP, it discards both the LSTM layer and the MLP altogether, using only a single linear layer inside each prediction head. All outputs are predicted directly from the shared LSTM layer backbone via a single linear projection per head, collapsing depth to reduce computation and parameters.

3.3 Training Procedure

Data Preprocessing. We employ distinct pipelines for the LSTM and Transformer architectures. In each pipeline, we begin by computing three (normalized) temporal features: duration (length of each activity), waiting time (interval between activities), and remaining time until case completion.

We prepend each case with two special events, “start” and “end”, before we compute and normalize its time-based features. The ”start” event marks the first activity from which the Transformer begins predicting what comes next, and the ”end” event tells the model that the case has finished. Finally, similar to the MTLFormer [7], all models are trained on prefixes of each case, ranging from length 1 to $n-1$, where n denotes the total number of events in the case. To build the training data, we extract every possible prefix of the chosen feature sequence: preceding activities for activity prediction, preceding roles for role

prediction, or preceding time features for timestamp prediction. In addition, we pad each prefix to the length of the longest case so it fits the Transformer's input size. Each padded prefix becomes the model input, and the very next feature in the sequence (the following activity, role, or time feature) serves as the target.

For the LSTM models, we adopt the preprocessing from Camargo et al. [5], which closely mirrors the Transformer pipeline but replaces prefix padding with n-gram construction. The only modification we introduce is to unify the normalization step: both pipelines now use the same z-score parameters. In practice, this means each numerical feature is scaled using the identical mean and standard deviation across both the LSTM and Transformer approaches.

Loss Functions. During training, we compute a separate loss for each head, that is, one for next-activity prediction, one for next-role prediction, and one for the three time features, resulting in three concurrent losses. Although this third head outputs three continuous time-related values rather than a single scalar, it still counts as one head, and we compute MSE jointly over all three predictions. Cross-entropy loss is applied to the categorical tasks (activity and role), while MSE loss governs the continuous time-feature predictions. To balance these multitask objectives, we employ uncertainty weighting [19] to combine them into a single scalar training loss.

Hyperparameter Settings. We performed separate grid searches optimized for Transformer-based and LSTM-based models.

For the Transformer-based models we explored embedding dimensions of 16 or 32 (ensuring divisibility by the number of heads), one, two or four attention heads, feed-forward dimensions of 32, 64 or 128, a fixed block dropout of 0.1, learning rates of 3e-4 or 6e-4, batch sizes of 8, 16 or 32, and one, two or four transformer encoder layers.

In contrast, for the LSTM models we varied learning rates across 5e-4, 1e-3, 5e-3, 3e-4 and 6e-4, batch sizes between 8, 16, 32 and 64, n-gram sizes of 5, 10 and 15, hidden layer sizes of 50, 25 and 10, a fixed tanh activation, thus ensuring a thorough and differentiated hyperparameter exploration for both architectures.

4 Evaluation

This section outlines the experiments conducted to evaluate the performance of the different Transformer and LSTM models. In the remainder, Section 4.1 describes the experimental setup, followed by the results in Section 4.2. The implementation can be found in our repository[1].

4.1 Experimental Setup

Datasets. Our evaluation is based on 5 event logs[2], spanning domains such as financial services, procurement, and manufacturing. As detailed in Table 1, these

[1] https://github.com/amaanansari/simplified-ppm-models.git
[2] Datasets available at https://zenodo.org/records/5734443

Table 1: Characteristics of the event logs.

Event Log	Type	#Traces	#Events	#Activities	#Resources
Production	real	225	4503	24	41
BPIC2012W	real	8616	59302	6	52
P2P	syn	608	9119	21	27
Confidential 1000	syn	1000	38160	42	14
Confidential 2000	syn	2000	77418	42	14

logs differ significantly in key properties such as the number of traces, events, activities, and resources. Crucially, all datasets contain both start and end timestamps for each event, allowing us to distinguish between the prediction tasks of next event time and next waiting time. Each dataset is sorted chronologically and then split horizontally into training, validation, and test sets, comprising 70%, 10%, and 20% of the cases, respectively.

Evaluation Metrics . To assess predictive performance, we employ F_1-score for the categorical tasks of next-activity and next-role prediction. For the continuous time-related tasks (waiting time, activity duration, and remaining time), we report the mean absolute error (MAE) in days.

Model Selection. Let $\mathcal{C}$ be the set of all candidate models produced by our hyperparameter grid, where each $M \in \mathcal{C}$ denotes a specific architecture–hyperparameter configuration. Let $M^\star = \arg\min_{M\in\mathcal{C}} \mathcal{L}(M)$ be the model with the lowest validation loss. For any $M \in \mathcal{C}$, define

$$p_M = \frac{\#\text{params}(M)}{\#\text{params}(M^\star)}, \qquad \ell_M = \frac{\mathcal{L}(M) - \mathcal{L}(M^\star)}{\mathcal{L}(M^\star)}.$$

Here, $\#\text{params}(M)$ denotes the number of *trainable* parameters of M, and $\mathcal{L}(M)$ is the validation objective (the same loss used during training) computed on the validation split. We define the composite score $S_M = p_M + \lambda \ell_M$, and select the model with the smallest score, $\arg\min_{M\in\mathcal{C}} S_M$. We set $\lambda = 2$ unless stated otherwise to emphasize validation performance relative to parameter savings. We apply this scoring procedure independently within each model type (e.g., $\text{MTLFormer}_{\text{light}}$, $\text{Transformer}_{\text{simple}}$, LSTM, $\text{LSTM}_{\text{light}}$) and for each dataset, with one model selected per type and dataset.

4.2 Results

Across five diverse event logs, we observe for both Transformer and LSTM models that their respective simplified variants achieve an enormous reduction in parameter count, on average over 80% fewer parameters, while only incurring minimal losses in predictive performance.

Looking at the Transformer-based architectures in more detail (see Table 2), across five diverse event logs, $\text{MTLFormer}_{\text{light}}$ reduces the number of parameters by 85% (from 136,412 to 19,823) compared to the full MTLFormer, while

Table 2: Results for the Transformer models for next activity prediction (NAP), next role prediction (NRP), next wait time prediction (NWTP), next duration prediction (NDP), and remaining time prediction (RTP).

Log	Model	NAP ↑	NRP ↑	NWTP ↓	NDP ↓	RTP ↓	Parameters
P2P	MTLFormer	0.84	0.93	3.32	0.11	24.41	93087
	$\text{MTLFormer}_{\text{light}}$	0.84	0.94	3.74	0.12	25.22	9519
	$\text{Transformer}_{\text{simple}}$	0.83	0.92	3.68	0.12	24.87	8407
BPI12W	MTLFormer	0.73	0.41	2.24	0.02	17.27	133 042
	$\text{MTLFormer}_{\text{light}}$	0.74	0.40	2.25	0.01	17.34	12514
	$\text{Transformer}_{\text{simple}}$	0.67	0.36	2.28	0.01	17.33	7954
Prod.	MTLFormer	0.21	0.33	1.53	0.2	37.03	169 003
	$\text{MTLFormer}_{\text{light}}$	0.15	0.29	1.56	0.20	36.66	26 555
	$\text{Transformer}_{\text{simple}}$	0.14	0.30	1.61	0.20	37.87	9823
C.1000	MTLFormer	0.86	0.93	0.04	0.04	0.98	151 847
	$\text{MTLFormer}_{\text{light}}$	0.84	0.90	0.04	0.04	0.99	27 175
	$\text{Transformer}_{\text{simple}}$	0.81	0.91	0.04	0.04	1.00	28 163
C.2000	MTLFormer	0.86	0.93	0.03	0.03	0.92	135079
	$\text{MTLFormer}_{\text{light}}$	0.86	0.92	0.03	0.04	0.94	23 351
	$\text{Transformer}_{\text{simple}}$	0.85	0.92	0.03	0.04	0.94	46 975
Average	MTLFormer	0.70	0.71	1.43	0.08	16.12	136 412
	$\text{MTLFormer}_{\text{light}}$	0.69	0.69	1.52	0.08	16.23	19 823
	$\text{Transformer}_{\text{simple}}$	0.66	0.68	1.53	0.08	16.40	20 264

experiencing a mere 1.4% decrease in the next-activity F_1 score (NAP, 0.70 → 0.69 F_1) and a 2.8% decrease in the next-role F_1 score (NRP, 0.71 → 0.69 F_1). Mean absolute errors for next wait time prediction (NWTP) increased by just 6.3%; next duration prediction (NDP) remained unchanged; and remaining time prediction (RTP) rose by only 0.7%. Meanwhile, $\text{Transformer}_{\text{simple}}$, which not only reduces the parameter count (136,412 → 20,264) but also simplifies the overall architecture, largely matches $\text{MTLFormer}_{\text{light}}$ on the time-related prediction tasks (NWTP MAE +7.0%, NDP unchanged, RTP MAE +1.7%). However, its next-activity prediction is, on average, about 3 percentage points lower (0.66 vs. 0.69 F_1), a drop driven primarily by the larger gap on BPI12, the biggest dataset in our evaluation. Overall, these results suggest that a single transformer encoder can suffice for time-related predictions, though the more elaborate $\text{MTLFormer}_{\text{light}}$ still holds a slight edge in next-activity accuracy.

Looking at the LSTM in Table 3, the $\text{LSTM}_{\text{light}}$ model reduces its larger variant by 77% (75,876 → 17,193 parameters), while incurring only minimal losses for the categorical prediction tasks. More specifically, we see a 2.9% drop

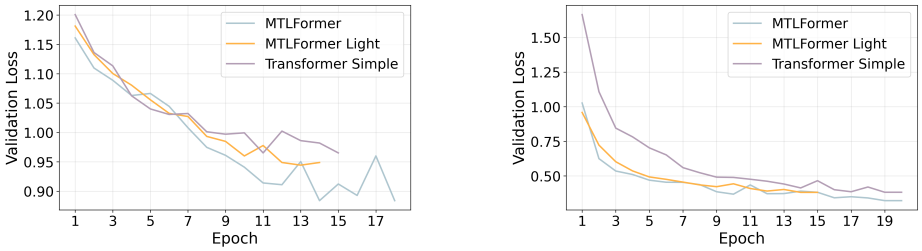
Table 3: Results for the LSTM models for next activity prediction (NAP), next role prediction (NRP), next wait time prediction (NWTP), next duration prediction (NDP), and remaining time prediction (RTP).

Log	Model	NAP ↑	NRP ↑	NWTP ↓	NDP ↓	RTP ↓	Parameters
P2P	LSTM	0.83	0.92	2.99	0.14	34.86	75 706
	$\text{LSTM}_{\text{light}}$	0.81	0.88	3.55	0.16	36.16	20431
BPI12	LSTM	0.71	0.39	1.99	0.02	23.04	74 574
	$\text{LSTM}_{\text{light}}$	0.68	0.38	2.09	0.02	23.48	19 824
Prod.	LSTM	0.25	0.36	1.89	0.2	40.8	76 808
	$\text{LSTM}_{\text{light}}$	0.20	0.31	2.1	0.2	42.02	4368
C.1000	LSTM	0.87	0.92	0.03	0.04	1.16	76 146
	$\text{LSTM}_{\text{light}}$	0.86	0.92	0.03	0.05	1.2	20 671
C.2000	LSTM	0.86	0.92	0.02	0.04	1.1	76 146
	$\text{LSTM}_{\text{light}}$	0.86	0.91	0.03	0.04	1.15	20 671
Avg.	LSTM	0.70	0.70	1.38	0.09	20.19	75 876
	$\text{LSTM}_{\text{light}}$	0.68	0.68	1.56	0.09	20.80	17 193

in both next-activity ($0.70 \rightarrow 0.68$ F_1) and next-role ($0.70 \rightarrow 0.68$ F_1) prediction. However, time-related errors are more affected: the next wait time MAE increases by 13%, the remaining time MAE by 3%, and the next duration MAE remains unchanged.

Compared to $\text{Transformer}_{\text{simple}}$, the $\text{LSTM}_{\text{light}}$ uses 15% fewer parameters (17,193 vs 20,264) and slightly improves next-activity prediction (0.68 vs 0.66 F_1), while matching next-role accuracy (0.68 F_1). However, it shows notably worse timing performance, most strikingly in remaining time prediction, with a 26.8% higher error (20.80 vs 16.40 MAE). Thus, while $\text{LSTM}_{\text{light}}$ is slightly more compact, $\text{Transformer}_{\text{simple}}$ offers superior temporal accuracy.

Furthermore, we analyze how the differently sized models compare in terms of validation loss progression. As shown in Figure 2, the validation-loss curves for BPIC2012W and P2P reveal that the simplified variants converge at a similar pace than their full-sized counterparts. Remarkably, despite a significantly reduced parameter count, $\text{MTLFormer}_{\text{light}}$ often reaches its minimum validation loss in fewer epochs than the original MTLFormer. This suggests that model compactness does not hinder, and may even enhance, the optimization process in these settings. We observed similar convergence behavior across all other event logs, underscoring the robustness of the lightweight architectures.



(a) Dataset: BPIC2012W (b) Dataset: P2P

Fig. 2: Validation loss curves for the three Transformer models.

5 Conclusion

In this work, we have investigated the feasibility of simplifying state-of-the-art deep learning architectures for Predictive Process Monitoring (PPM), aiming to reduce computational overhead and, at the same time, preserve predictive performance. In summary, our experiments across five diverse event logs show that deep PPM models can be significantly compressed without a proportional loss in predictive performance. Pruning both LSTM- and Transformer-based architectures, we found Transformers to be particularly robust: $\text{MTLFormer}_{\text{light}}$, which preserves the architecture of its larger variant while reducing parameters by 85%, retains 98–99% of the original F_1 performance and shows only minimal increases in MAE for time predictions. By reducing the architecture to a single attention encoder, $\text{Transformer}_{\text{simple}}$ still retains 94–96% of the original F_1 performance, demonstrating that even this streamlined design can compete with more complex models in PPM tasks. Moreover, $\text{MTLFormer}_{\text{light}}$ often converges more quickly during training than its larger counterpart, suggesting that compactness may facilitate the optimization process in certain cases.

In contrast, pruning the LSTM incurs more significant performance penalties, highlighting a fundamental difference in how these architectures leverage parameter capacity. These insights underscore the potential of lightweight attention models as the backbone for PPM systems.

Future work could explore architecture-aware pruning techniques or neural architecture search to further optimize the trade-off between efficiency and accuracy. Most PPM tasks in this study are limited to next-event prediction; exploring more longer-term targets, such as suffix or outcome prediction, could provide valuable insights. Additionally, investigating the generalizability of lightweight models across unseen domains and their integration into real-time PPM systems offers promising directions for practical deployment.

References

1. Di Francescomarino, C., Ghidini, C.: Predictive process monitoring. In: Process mining handbook. Springer International Publishing Cham (2022) 320–346

2. Rama-Maneiro, E., Vidal, J.C., Lama, M.: Deep learning for predictive business process monitoring: Review and benchmark. IEEE Transactions on Services Computing **16**(1) (2021) 739–756
3. Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M.: Outcome-oriented predictive process monitoring: Review and benchmark. ACM Transactions on Knowledge Discovery from Data (TKDD) **13**(2) (2019) 1–57
4. Tax, N., Verenich, I., Rosa, M.L., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: CAiSE. Volume 10253 of Lecture Notes in Computer Science., Springer (2017) 477–492
5. Camargo, M., Dumas, M., Rojas, O.G.: Learning accurate LSTM models of business processes. In: BPM 2019, Vienna, Austria, September 1-6, 2019, Proceedings. Volume 11675 of Lecture Notes in Computer Science., Springer (2019) 286–302
6. Bukhsh, Z.A., Saeed, A., Dijkman, R.M.: Processtransformer: Predictive business process monitoring with transformer network. CoRR **abs/2104.00721** (2021)
7. Wang, J., Huang, J., Ma, X., Li, Z., Wang, Y., Yu, D.: Mtlformer: Multi-task learning guided transformer network for business process prediction. IEEE Access **11** (2023) 76722–76738
8. Pandey, S., Nepal, S., Chen, S.: A test-bed for the evaluation of business process prediction techniques. In: CollaborateCom. (2011) 382–391
9. Folino, F., Guarascio, M., Pontieri, L.: Context-aware predictions on business processes: An ensemble-based solution. In: New Frontiers in Mining Complex Patterns, Berlin, Heidelberg, Springer Berlin Heidelberg (2013) 215–229
10. Rogge-Solti, A., Weske, M.: Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In: Service-Oriented Computing, Berlin, Heidelberg, Springer Berlin Heidelberg (2013) 389–403
11. Conforti, R., de Leoni, M., La Rosa, M., van der Aalst, W.M.P.: Supporting risk-informed decisions during business process execution. In: CAiSE. CAiSE'13, Berlin, Heidelberg, Springer-Verlag (2013) 116–132
12. Evermann, J., Rehse, J., Fettke, P.: Predicting process behaviour using deep learning. Decis. Support Syst. **100** (2017) 129–140
13. Amiri Elyasi, K., van der Aa, H., Stuckenschmidt, H.: Pgtnet: A process graph transformer network for remaining time prediction of business process instances. In: CAiSE, Berlin, Heidelberg, Springer-Verlag (2024) 124–140
14. Özdemir, K., Kirchdorfer, L., Elyasi, K.A., Aa, H.v.d., Stuckenschmidt, H.: Rethinking business process simulation: A utility-based evaluation framework. In: BPM, Springer (2025) 128–144
15. Fani Sani, M., Vazifehdoostirani, M., Park, G., Pegoraro, M., van Zelst, S.J., van der Aalst, W.M.P.: Performance-preserving event log sampling for predictive monitoring. J. Intell. Inf. Syst. **61**(1) (March 2023) 53–82
16. Pauwels, S., Calders, T.: Incremental predictive process monitoring: The next activity case. In: BPM, Berlin, Heidelberg, Springer-Verlag (2021) 123–140
17. Weytjens, H., De Weerdt, J.: Learning uncertainty with artificial neural networks for predictive process monitoring. Applied Soft Computing **125** (2022) 109134
18. Roider, J., Zanca, D., Eskofier, B.M.: Efficient training of recurrent neural networks for remaining time prediction in predictive process monitoring. In: BPM, Berlin, Heidelberg, Springer-Verlag (2024) 238–255
19. Kendall, A., Gal, Y., Cipolla, R.: Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In: CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018, IEEE Computer Society (2018) 7482–7491